Synthetic Aperture Beamformation using the GPU

Hansen, Jens Munk; Schaa, Dana; Jensen, Jørgen Arendt

Published in:
Proceedings of IEEE International Ultrasonics Symposium

Link to article, DOI:
10.1109/ULTSYM.2011.0089

Publication date:
2011

Document Version
Early version, also known as pre-print

Paper presented at the IEEE International Ultrasonics Symposium, Orlando, Florida, 2011:

Synthetic Aperture Beamformation using the GPU

Jens Munk Hansen[†], *Dana Schaa*[⋆] *and Jørgen Arendt Jensen*[†]

[†]Center for Fast Ultrasound Imaging,
Biomedical Engineering group, Department of Electrical Engineering, Bldg. 349,
Technical University of Denmark, DK-2800 Kgs. Lyngby, Denmark

[⋆]Dept. of Electrical and Computer Engineering,
Northeastern University, Boston, MA, USA

To be published in Proceedings of IEEE International Ultrasonics Symposium, Orlando, Florida, 2011.

Synthetic Aperture Beamformation using the GPU

Jens Munk Hansen[†], Dana Schaa[⋆] and Jørgen Arendt Jensen[†].
[†]Center for Fast Ultrasound Imaging, Dept. of Elec. Eng,
Technical University of Denmark, DK-2800 Lyngby, Denmark
[⋆] Dept. of Electrical and Computer Engineering,
Northeastern University, Boston, MA, USA

***Abstract*—A synthetic aperture ultrasound beamformer is implemented for a GPU using the OpenCL framework. The implementation supports beamformation of either RF signals or complex baseband signals. Transmit and receive apodization can be either parametric or dynamic using a fixed F-number, a reference, and a direction. Images can be formed using an arbitrary number of emissions and receive channels. Data can be read from Matlab or directly from memory and the setup can be configured using Matlab. A large number of different setups has been investigated and the frame rate measured. A frame rate of 40 frames per second is obtained for full synthetic aperture imaging using 16 emissions and 64 receive channels for an image size of 512x512 pixels and 4000 complex 32-bit samples recorded at 40 MHz. This amount to a speed up of more than a factor of 6 compared to a highly optimized beamformer running on a powerful workstation with 2 quad-core Xeon-processors.**

I. Introduction

Image quality and diagnostic capabilities of medical imaging depend on the inversion of the measured data for the given modality. For ultrasound imaging, the inversion is primarily made by delay-and-sum beamformation. This comprises computation and application of channel delays and apodization for both the emissions and the individual receiving elements. Ideally, one would like the result of the beamformation to approximate the true inverse of the forward model, which itself is a complex model of both time and space. A forward model or simulation model is described by the ultrasound simulation program Field II [1], [2]. This may sound like elements of the future, but the evolution of ultrasound beamformers moving from analog into digital implementations, described by Thomenius [3], has made it possible to implement more advanced beamformers and new and improved methods of beamformation are still emerging. At the moment, one of the more demanding methods is synthetic aperture (SA) imaging [4] and variations hereof. Even though computers today are fast, the limiting factor for a SA ultrasound system is still the memory IO resources available.

An equally high demand for memory throughput is found in the computer gaming industry, where hundreds of megabytes of data are processed every second for rendering a scene in a 3D computer game. The processing takes place on the graphics processing unit (GPU), which is a many-core massively parallel throughput-oriented execution unit. It contains a lot of arithmetic logic units (ALUs) and is suitable for single-instruction-multiple-data (SIMD) execution. Until the fourth generation of GPUs, the GPUs were all fixed-function, but since then the vendors have introduced vertex-level and pixel-level programmability and several high-level graphics languages has been released, which allow programs written in C/C++ to use a runtime and load programs to be executed by the GPU. In this paper, the most recent framework, OpenCL [5] is used for SA beamformation of ultrasound data. Previous work has already done using multiple GPUs for SA beamformation of ultrasound data [6]. This work is different in the way that a more advanced apodization is used and the beamformer can be configured using Matlab.

II. GPU Hardware

The GPUs consist of several compute units, which each can be thought of as (a collection of) multi-core processors sharing some local memory and a common pipeline. This means in particularly that the groups of cores read and write data simultaneously and this should be kept in mind when implementing programs for the GPU to execute. Further, no caching or very little caching is done by the hardware, so memory handling is therefore more critical for GPU programs than for programs implemented for CPUs. The two manufacturers of GPUs, Nvidia and AMD/ATI have created two very different architectures for memory access on their devices. AMD operate with vectorized memory reads and writes and a uniform memory space exposition [7]. This is very similar to how SIMD is implemented on CPUs and the programmer has to think of organizing data as 128-bit vectors (4 floats) to achieve good performance. This makes the programming less flexible but potentially speed-up calculations involving vectorized input and output. Many programmers forget this and falsely arrive at a poor performance. Nvidia work instead with a two-level thread hierarchy and scalar memory addressing and does not take advantage of instruction level parallelism to the same extent as their rival [8]. The memory access pattern for synthetic aperture beamformation is randomly shifted and very few arithmetic operations are performed compared to load and store operations. It is therefore an advantage to have a large amount of L1 cache available for each compute unit. The latest generation GPUs from Nvidia, Fermi, have up to 48 kB of L1 cache available for each compute unit, while the Evergreen family of their rival AMD, only has 8 kB. Further, since beamformation of a single pixel using linear interpolation involves only two consecutive values of RF data for each channel, you do not directly benefit from vectorized memory reads, but often more advanced interpolation is needed which can benefit

from this way of adresseing memory. These considerations together with some initial performance measurements, made us focus on the Nvidia architecture.

A. SIMD cores

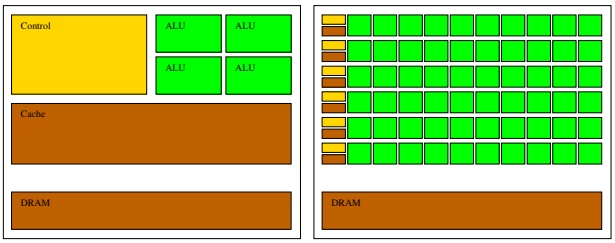
The GPU primarily used in this article is the GTX-580 from Nvidia. It has 16 compute units or streaming multiprocessors (SMs), each containing 2 groups of 16 streaming processors (SPs), 4 special function units (SFUs) and 16 load/store units. The SMs have a SIMD architecture. Scalar threads are grouped into SIMD groups called *warps*, with 32 scalar threads per warp. Each SP can execute a sequential thread, the SPs execute in what Nvidia calls SIMT (Single Instruction, Multiple Thread) fashion; all SPs in the same group execute the same instruction at the same time, much like classical SIMD processors. SIMT handles conditionals somewhat differently than SIMD, though the effect is much the same, where some cores are disabled for conditional operations.

The SM double pumps each group of 16 SPs to execute one instruction for each of two warps in two clock cycles, for integer or single-precision floating point operations. For double-precision instructions, the SM combines the two groups of cores to look like a single 16-core double-precision multiprocessor; this means the peak double-precision throughput is 1/2 of the single-precision throughput.

Another important feature of the GPUs is how multithreading is designed to hide memory and pipeline latencies. To facilitate low-cost context switching, all simultaneously running threads on an SM keep their register states in the same resister file. The number of registers consumed by a thread depends on the program and it is possible to create more threads than what can fit simultaneously in the register file, but the user should avoid this. In addition to the register file and L1 cache, there is also a small local memory storage on each SM called shared memory that is partitioned among groups of threads called thread blocks. This can be used by the programmer for explicit caching of data.

The scheduling of the threads takes place by a dual warp scheduler that can occupy both 16-wide groups of SPs with separate warps via dual issue. Each SM can track a total of 48 warps simultaneously and schedule them pretty freely in intermixed fashion, switching between warps at will from one cycle to the next. Obviously, this should be a very effective means of keeping the execution units busy, even if some of the warps must wait on memory accesses, because many other warps are available to run. To give an idea of the scale involved, consider 32 threads times 48 warps per SM. This adds up to 25,576 concurrent threads in flight on the GPU.

This approach for keeping execution units busy is much simpler than what goes on in a modern CPU, where a larger instruction set is available, caching is done at multiple levels, and branch prediction is used to improve the flow in the instruction pipeline. In Fig 1, an illustration is given of how much area is used for control logic, ALUs, and caches in a GPU compared to what is used in a modern quad-core CPU.



(a) CPU (b) GPU

Figure 1. Simplified hardware layout for a quad-core CPU and a GPU with 60 SPs arranged in groups of 10.

Finally, we should mention that the previous Tesla generation has up to 30 SMs (GTX-285), but each SM contain only a single group of 8 SPs and the cores in this group are quad-pumped to execute one instruction for an entire warp, 32 threads, in four clock cycles. In addition to the 8 SP, each SM contain a shared SFU, which handle transcendentals and double-precision operations at 1/8 the compute bandwidth. This is four times slower than the new Fermi architecture.

B. Memory IO

As stated in the introduction, beamformation of ultrasound data comprises computation and application of channel delays and apodizations. The former also referred to as focusing amounts to massive distance calculations followed by memory look-ups. The apodization part consists of a weighting based on the pixel location and the origin of the ultrasound signals. The amount of calculations are massive, but the memory access pattern is randomly shifted. This is suboptimal for the current GPU architectures and the result is a heavily memory bound application. Because of this, to achieve a high performance, a high memory bandwidth as well as low latency for access to the global memory is preferable. The main memory on the GTX-580 delivers 192 gigabytes per second (GB/s). This is six times the bandwidth of a Core-i7 with triple-channel DDR3-1333 memory, delivering 32 GB/s. The memory speedup is obtained using multiple 64-bits interfaces (6 instead of 3) and higher clock values. The latency for accessing the global memory is several hundred clock cycles so a high bandwidth alone is not sufficient for a high performance. Multithreading and caching done by the programmer or by the hardware is used for hiding latency and keeping a high memory throughput. The Fermi also features a 768 kB unified L2 cache that services all load, store, and texture requests. The L2 provides efficient, high speed data sharing across the GPU. Algorithms for which data addresses are not known beforehand, like beamformation benefit from the cache hierarchy. Filter and convolution kernels that require multiple SMs to read the same data also benefit. Members of the AMD Evergreen family only contain 128 kB L2 cache.

III. Design

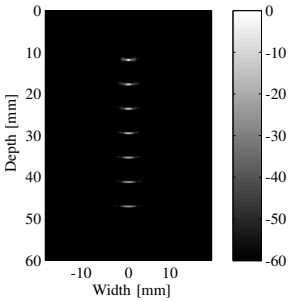
To cover as many focusing strategies as possible and thereby be able to make important conclusion on processing capabilities and throughput, a full synthetic aperture (SA)

focusing system is implemented with possibilities for later simplifications to improve on the performance. In addition to this choice, the following decisions were made with respect to design and functionality of the beamformer:

- The implementation should support beamformation of either RF signals or complex base band signals.
- The input data should be read directly from memory or from `.mat` files supported by Matlab. In this way, data simulated with Field II as well as data acquired using our research scanner SARUS [9] can be processed.
- Support for full parametric or dynamic apodization using using a fixed F-number, a reference point, and a direction.
- Beamformation should support an arbitrary number of emissions and receive channels.
- Parameters used for beamformation should be stored in configuration files, which are read once before processing starts with a given setup.
- The frame rate should be measured for continuous beamformation and display using a fixed setup.

IV. Validation

To verify the correctness of the implementation, synthetic aperture RF data were simulated with Field II for 7 scatterers located on a line perpendicular to the transducer surface and passing through the center position of the transducer. IQ data were formed using Hilbert transformation and data were initially beamformed using BFT3 - a Matlab toolbox written in C++ [10]. The resulting image served as a reference for the output of the GPU beamformed. Next, the data were beamformed using a simple program written in ANSI-C, which later was used as a reference for debugging the GPU kernels. Initially, no apodization was included to focus only on the correctness of the delay calculation and interpolation. Later, dynamic receive apodization using a Hamming window and an F-number of 1.0 was introduced. The parameters used for the simulation and resulting image are shown in Fig 2



Parameter	value
Sampling frequency	120 MHz
# elements	16
# emissions	16
# samples	4000 complex
Size	7.8 Mbyte

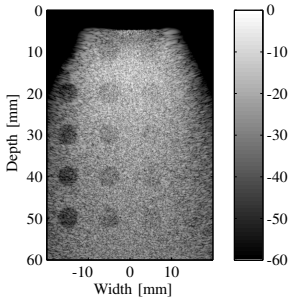
Figure 2. 7 scatter phantom image and parameters used for simulation. The image is beamformed using 1024 lines, each with 1024 samples.

V. Implementation

In the OpenCL framework, one operates with *work-groups* which are groups of threads associated with pixels in the resulting image. Several concurrent work-groups can reside on one compute unit depending on the work-group's memory requirements and compute unit's memory resources. The optimal dimensions of the work-groups therefore depend on the registers and memory needed for the calculations, the amount of ALUs required for the calculations themselves, and the amount of IO to the global memory. The initial implementation was made by starting with an OpenCL kernel reproducing the results of the simple ANSI-C program applied to the RF data for the 7 scatter phantom. The dimension of the work-group was adjusted to achieve the best performance for this setup. Apodization was introduced and by experimenting with doing some calculations on a per work-item and some on a per work-group basis, no effect was seen on the performance confirming that the application is memory bound.

VI. Optimization

Moving on to a larger dataset resembling what is actually acquired using a synthetic aperture approach. RF data for a cyst phantom was simulated for 192 channels and 16 emissions. The image and parameters used for simulation is given in Fig. 3 The total execution for synthetic aperture imaging



Parameter	value
Sampling frequency	40 MHz
# elements	192
# emissions	16
# scatterers	500.000
# samples	4000 complex
Size	93.8 Mbyte

Figure 3. Cyst phantom image and parameters used for simulation. The image is beamformed using 1024 lines, each with 1024 samples.

of the 7 scatterers and cyst phantom using dynamic transmit and receive apodization is given in Table I. The execution time for beamformation using the BFT3 toolbox and a faster SIMD implementation on a dual CPU workstation is also included. We note that for this kernel, the GTX-285 is much faster than the more expensive GPU from AMD, HD 5870. Several

Table I

Phantom	GPU				2× CPU, E5520, 2.26 GHz	
	GTX-580	GTX-480	GTX-285	HD 5870	SIMD	BFT3
7 scatter	0.087 sec	0.089 sec	0.28 sec	N/A	N/A	N/A
Cyst	0.46 sec	0.48 sec	0.92 sec	2.3 sec	5.48 sec	15.9 sec

attempts for optimization was made including:

- Using interleaved complex data rather than split format. In this way, the real and imaginary part of the relevant samples are next to each other memory-wise.
- Using the local memory for caching the samples used for beamformation the pixels in each work-group.
- Precalculating all delays and/or apodizations to keep calculations to an absolute minimum.

- Using the faster texture memory for loading a fraction of the RF data at a time for beamforming the image in multiple stages. The reason for this is that the texture is not big enough for data from 192 channels, each containing 4000 complex samples.

Using, the AMD HD 5870, we were only able to speed up the calculation by 15% and this was obtained by using interleaved data together with caching all samples used for a work-group in the local cache and accessing the samples using the faster cache rather than directly in the slower global memory. The precalculation of delays gave no speedup, which was expected since the application is heavily memory bound. We were not able to achieve any improvements using the local memory, but this deserves more attention.

VII. Results

After having played around with different optimizations using the AMD HD 5870 and the GTX-285 from Nvidia, the latest generation of GPUs from Nvidia, here represented by GTX-480 and GTX-580 were tested. Using pinned or page-locked memory for the input data, it was possible to do overlapped IO such that while the GPU was busy beamforming data from one emission, data for the following emission was being copied. Since, a sum of contributions is computed from a number of emissions and for a set emissions the image is read back once or continuously imaged on the screen, this application is ideal for overlapped IO. Unfortunately more time is spent on beamforming than copying data from the CPU host to the GPU device, so only about a 50% speed up was achieved using overlapped IO. Moreover, since we are beamforming IQ data, it is sufficient beamforming a smaller image, since the frequency content of the enveloped signals is of lower values. Using the GTX-580 GPU, a work-group size of 32x32 pixels, the frame rates listed in Table II were obtained

Table II
Frame rates for an image size of 512x512 pixels beamformed using 4000 complex samples for each receive channel. The frame rates include reading back each frame to the CPU.

		# emissions			
		4	8	16	32
# channels	192	42.8	23.7	12.5	7.2
	128	70.4	39.9	20.0	11.2
	96	77.6	44.8	23.8	13.3
	64	137.3	75.4	39.7	21.1
	48	171.3	80.5	49.7	26.2
	24	226.2	143.7	86.1	44.2

VIII. Discussion

The motivation for the implementation of the SA beamformed for this article was to speed-up development of new beamformers rather than making a good enough solution which can easily be adopted for a commercial implementation. However, the results show that even with a naive kernel, where only little work has been done for optimization, decent frame rates can be obtained. Later experiments, where 16-bit samples was used instead of 32-bit single precision floating point data, revealed an additional speed-up of a factor of two. Other attempts that could be made include; computing time-of-flight and apodization using look-up-tables combined with piecewise linear continuation.

Another way of speeding up the beamformation is to use multiple GPUs and this will obviously work, since the PCI-X 2.0 x16 delivers easily 5 GB/s and the data rate for the fastest setup in this article is still at least a factor of ten below this value.

IX. Conclusion

In this article, proof-of-concept is given for synthetic aperture beamformer running on the GPU supporting dynamic transmit and receive apodization. Experiments show that a naive beamformer performs much better on Nvidia Fermi GPUs than on the AMD Evergreen GPUs using the OpenCL 1.0 framework. Having studied the little information the vendors give on their hardware, the better performance on the Nvidia GPUs for naive kernels is most likely due to their two-level thread hierarchy and larger amount of L1 and L2 caches.

References

[1] J. A. Jensen and N. B. Svendsen, "Calculation of Pressure Fields from Arbitrarily Shaped, Apodized, and Excited Ultrasound Transducers," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 39, pp. 262–267, 1992.

[2] J. A. Jensen, "Field: A Program for Simulating Ultrasound Systems," *Med. Biol. Eng. Comp.*, vol. 10th Nordic-Baltic Conference on Biomedical Imaging, Vol. 4, Supplement 1, Part 1, pp. 351–353, 1996.

[3] K. E. Thomenius, "Evolution of ultrasound beamformers," in *Proc. IEEE Ultrason. Symp.*, vol. 2, 1996, pp. 1615–1621.

[4] G. R. Lockwood, J. R. Talman, and S. S. Brunke, "Real-time 3-D ultrasound imaging using sparse synthetic aperture beamforming," *IEEE Trans. Ultrason., Ferroelec., Freq. Contr.*, vol. 45, pp. 980–988, 1998.

[5] K. O. W. Group, *The OpenCL Specification, version 1.0.29*, 8 December 2008. [Online]. Available: http://khronos.org/registry/cl/specs/opencl-1.0.29.pdf

[6] B. Y. S. Yiu, I. K. H. Tsang, and A. C. H. Yu, "Real-time GPU-based software beamformer designed for advanced imaging methods research," in *Proc. IEEE Ultrason. Symp.*, 2010, pp. 1920–1923.

[7] A. M. Devices, "Heterogeneous computing. OpenCL™and the ATI Radeon™ HD 5870 ("Evergreen") architecture," 2010.

[8] *Whitepaper NVIDIA's Next Generation CUDA Compute Architecture: Fermi, v 1.1*.

[9] H. Holten-Lund, I. Nikolov, and M. Hansen, "SARUS digital acquisition and ultrasound processing board, processing specification," Ørsted•DTU, Technical University of Denmark and Prevas A/S, Tech. Rep., 2007.

[10] J. M. Hansen, M. C. Hemmsen, and J. A. Jensen, "An object-oriented multi-threaded software beam formation toolbox," in *Proc. SPIE - Medical Imaging - Ultrasonic Imaging and Signal Processsing*, vol. 7968, 2011, p. 79680Y.